Reprisé et Compléments au TD08 : autre implémentation de l'algorithme

1. Construire la représentation

Fonction : make-individu

```
;;1. si le frame concept n'exist pas erreurs
;;2. creer un id unique
;;3. creer un début de frame avec le nom et individu
;;4. pour chaque slot .vérifier que le slot est autorisé  .si oui i faut créer slot
avec la valeur correspondante est ajouter au début de frame précédent
;;5. ajouter id à *frame*
;;6. retourne id
```

```
(defun make-individuP (name concept &rest prop-val)
    (unless (member concept *frames*) (error "Le concept n'existe pas"))
    (let ((id (gentemp "F")) (allowed-slots (mapcar 'car (cdr (symbol-value concept)))) slot value fn)
        (set id (list name (list 'type (list 'value 'individu)) (list 'is-a (list 'value concept))))
        (loop
            (unless prop-val (return NIL))
            (setq slot (pop prop-val))
            (setq value (pop prop-val))
            (when (member slot allowed-slots)
                (setq fn (cadr (assoc 'if-added (cdr (assoc slot (cdr (symbol-value concept)))))))
                (if fn
                    (setq value (funcall fn slot value))
                )
                ;; (set value (if fn (funcall fn slot value)))
                (when value
                    (set id (append (symbol-value id) (list (list slot (list 'value value)))) )
                )
            )
        )
        (pushnew id *frames*)
    )
)
```

- PS : C'est la version du Monsieur Lenne. Pendant mon essayage, il y a une petite erreur dans line vert : (set value (if fn (funcall fn slot value))).
  - Il faut tout d'abord tester si fn exist ou pas et puis

    (setq value (funcall fn slot value))

- Test et Résultat :

```
(make-individuP 'Clyde 'ELEPHANT 'COLOR 'blue 'AGE 5)
(make-individuP 'Clyde2 'ELEPHANT 'AGE 7)
(print *frames*)
(print F1)
(print F2)
```

```
(F2 F1 ELEPHANT)
(CLYDE (TYPE (VALUE INDIVIDU)) (IS-A (VALUE ELEPHANT)) (COLOR (VALUE BLUE)) (AGE
(VALUE 5)))
(CLYDE2 (TYPE (VALUE INDIVIDU)) (IS-A (VALUE ELEPHANT)) (AGE (VALUE 7)))
```

- Fonction get-slot-value

```
(defun get-slot-value (frame slot)
  ; frame : nom du frame
  ; slot : nom du slot
    (unless (member frame *frames*) (error "Le concept ~S n'est pas"))
    (let ((get-slots (cdr (symbol-value frame))) val ) ; on récupère toutes les slots
      (if (setq val (cadr (assoc 'value (cdr (assoc slot get-slots))))) ; on regarde si la valeur est présente
          val
          (get-slot-value-inherit (cadr (assoc 'value (cdr (assoc 'is-a get-slots)))) slot) ; sinon on regarde si elle
peut être héritée
      )
    )
)

(defun get-slot-value-inherit (frame slot)
```

```
  ; on cherche une valeur hérité du slot présent le frame
  ; frame : nom du frame
  ; slot : nom du slot
    (if (not (member frame *frames*))
        nil ;arrêt de la récursion si l'on ne trouve pas
        (let ((props (cdr (symbol-value frame))) val)
          (if (setq val (cdr (assoc 'default (cdr (assoc slot props))))) ; on regard si la valeur est présente par
défaut
              val
              (get-slot-value-inherit (cadr (assoc 'value (cdr (assoc 'is-a props)))) slot)
          )
        )
    )
) ; sinon on remonte
```

- Test et Résultat :

```
•                   (print (get-slot-value 'F1 'COLOR) )
•                   (print (get-slot-value 'F1 'AGE) )
•                   (print (get-slot-value 'F2 'COLOR) )
```

```
YanLIUdeMacBook-Pro:Lisp yann$ clisp TD08.lisp
(F2 F1 ELEPHANT)
(CLYDE (TYPE (VALUE INDIVIDU)) (IS-A (VALUE ELEPHANT)) (COLOR (VALUE BLUE)) (AGE (VALUE 5)))
(CLYDE2 (TYPE (VALUE INDIVIDU)) (IS-A (VALUE ELEPHANT)) (AGE (VALUE 7)))
BLUE
5
GREY      →Default
YanLIUdeMacBook-Pro:Lisp yann$
```

TD09 : Représentation Objet

1. Comment représenter un objet ? Nous prendrons l'exemple d'Albert, qui est agé de 26 ans et qui a deux frères : Max et Jean.

```
(Albert
  (AGE . 26)
  (FRERES . (Max Jean))
)
(Max
  (FRERES . (Albert))
)
(Jean
```

```
  (FRERES . (Albert))
)
```

Proposition en TD :

```
$0 : ( (NAME "Albert") (AGE 26) (FRERE $1 $2) (TYPE $PERSONNE) )
$1 : ( (NAME "Jean") (FRERE $0 $2) (TYPE $PERSONNE) )
$2 : ( (NAME "Max") (FRERE $0 $1) (TYPE $PERSONNE) )
```

2. Comment peut-on abstraire cet objet ?

On peut créer une classe :

```
$PERSONNE : ( (NAME "PERSONNE") (type $CLASS) (ATTR AGE NAME) (REL FRERES SOEURS) )
```

3. À partir de cette représentation, comment pouvons-nous déduire qu'un étudiant est une personne ? qu'il peut avoir des frères et des sœurs ?

On définit tout d'abord une classe $ETUDIANT

```
$ETUDIANT : ( (NAME "ETUDIANT") (IS-A $PERSONNE) (TYPE $class) (ATTR numCarte))
```

IS-A : du côté des classes
TYPE : pour les instanciations

On se réfère aux classes et on regarde si un étudiant est une personne : on intérroge *la représentation conceptuelle* (qui ne constitue pas * l'ensemble des instances réifiées du monde réel*).

```
$CLASSE : ((TYPE $CLASSE) (IS-A $OBJECT) (ATTR TYPE ATTR REL IS-A) (REL ...) )
```

4. Avec une telle représentation, comment peut-on introduire des valeurs par défaut au niveau des propriétés ?

On peut représenter les propriétés

```
$AGE : ((TYPE INTEGER) (DEFAULT 22))

-définir un individu "idéal"
 $default ((age 22) (nom "Nom")
-ou modaliser les propriétes
  $age ((type arrt) (default 22) (owner $person))
```

5. Comment peut-on exprimer des contraintes sur les objets d'une telle représentation (par exemple, le fait qu'une personne de sexe masculin ne peut être mère de quelqu'un) ?

On utilise des règles ou des méthodes (on n'est plus dans le déclaratif)

Bonus : Utilisation de mots clefs

&key : Permet de passer les paramètres dans l'ordre que l'on veut.

```
(defun F (a &key name)
..... name .....)

(F 5 :name "max")
```

&rest : permet de récupérer plus de paramètres

```
(defun G (a &rest prop) ...)
```

prop est une liste formée de l'ensemble des paramètres de G après a lors de l'appel de G.

```
(G a p1 p2 p3)
```

&optional : permet de rendre certains paramètres optionnels